On the overfly algorithm in deep learning of neural networks

Alexei Tsygvintsev

Version of Record: https://www.sciencedirect.com/science/article/pii/S0096300318311160
Manuscript_1a6b86d450787961f8f60828183cc1c8

.

ON THE OVERFLY ALGORITHM IN DEEP LEARNING OF NEURAL NETWORKS

ALEXEI TSYGVINTSEV

Abstract. In this paper we investigate the supervised backpropagation training of multilayer neural networks from a dynamical systems point of view. We discuss some links with the qualitative theory of differential equations and introduce the overfly algorithm to tackle the local minima problem. Our approach is based on the existence of first integrals of the generalised gradient system with build–in dissipation.

1. Introduction. The dynamics of gradient flow. Neural networks and backpropagation.

Let $F : U \to \mathbb{R}$ be a smooth function in some open domain $U \subset \mathbb{R}^n$. We equip U with the topology induced by the standard Euclidean norm $||\cdot||$ defined by the canonical scalar product $< x, y >= \sum x_i y_i$. The gradient vector field defined in U by F is given by $V(x) = -\nabla F = -(\frac{\partial F}{\partial x_1}, \ldots, \frac{\partial F}{\partial x_n})^T$, where $x = (x_1, \ldots, x_n)^T$ are canonical coordinates in U. The critical points of F are the solutions of $V(x) = 0$, $x \in U$. Let K be the set of all critical points of F in U (which can be unbounded and/or contain non–isolated points).

The following theorem [10], [19] is a classical result describing the asymptotic behaviour of solutions of the gradient differential system:

$$x' = V(x), \quad x \in U\,. \tag{1.1}$$

Theorem 1.1. *Let $x_0 \in U$ be the initial condition of (1.1). Then every solution $t \mapsto x(t)$, $x(0) = x_0$ either leaves all compact subsets of U or approaches as $t \to +\infty$ the critical set K i.e*

$$\lim_{t \to +\infty} \inf_{y \in K} ||x(t) - y|| = 0\,. \tag{1.2}$$

In particular, at regular points, the trajectories of (1.1) cross the level surfaces of F orthogonally and isolated minima of F (which is a *Lyapunov* function [14]) are asymptotically equilibrium points.

Under the additional analyticity condition the above convergence result can be made stronger:

Key words and phrases. deep learning, neural networks, dynamical systems, gradient descent.

Theorem 1.2. *(Absila,Kurdyka,* [3]*) Let F be real analytic in U. Then $y \in K$ is a local minimum of F iff it is asymptotically stable equilibrium point of* (1.1).

It should be noticed that the gradient system (1.1) can not have any non–constant periodic or recurrent solutions, homoclinic orbits or heteroclitic cycles. Thus, trajectories of gradient dynamical systems have quite simple asymptotic behaviour.

Nevertheless, the localisation of basin of attraction of any equilibrium point (stable or saddle one) belonging to K is a non trivial problem.

Supervised machine learning in multi–layered neural networks can be considered as application of gradient descent method in a non–convex optimization problem. The corresponding cost (or error) functions are of the general form

$$E = \frac{1}{2} \sum (p_i - f(W, A^i)^2 \,, \tag{1.3}$$

with data set (A^i, p_i) and a certain highly non–linear function f containing the weights W. The main problem of the machine learning is to minimize the cost function E with a suitable choice of weights W. A gradient method, described above and called *backpropagation* in the context of neural network training, can get stuck in local minima or take very long time to run in order to optimize E. This is due to the fact that general properties of the cost surface are usually unknown and only the trial and error numerical methods are available (see [4], [12], [16], [9], [17], [18], [5])). No theoretical approach is known to provide the exact initial weights in backpropagation with guaranteed convergence to the global minima of E. One of most powerful techniques used in backpropagation is the *adaptive learning rate selection* [8] where the step size of iterations is gradually raised in order to escape a local minimum. Another approach is based on *random initialization* [15] of weights in order to fortunately select them to be close to the values that give the global minimum of the cost function. The deterministic approach, called *global descent*, was proposed in [7] where optimization was formulated in terms of the flow of a special deterministic dynamical system.

The present work seeks to integrate the ideas from the theory of ordinary differential equations to enrich the theoretical framework and assist in better understanding the nature of convergence in the training of multi–layered neural networks. The principal contribution is to propose the natural extension of classical gradient descent method by adding new degrees of freedom and reformulating the problem in the new extended phase space of higher dimension. We argue that this brings a deeper insight into the convergence problem since new equation become simpler algebraically and admit a family of known first integrals. While this proposal may seem radical, we believe that it offers a number of

advantages on both theoretical and as numerical levels as our experiments clearly show. Common sense suggests that embedding the dynamics of a gradient flow in a more general phase space of a new more general dynamical system is always advantageous since it can bring new possibilities to improve the convergence and escape local minima by embedding the cost surface into the higher dimensional phase space.

The study is divided into three parts. In Section 2 we begin by reminding how the gradient descent method is applied to train the simplest possible neural network with only output layer. That corresponds to the conventional backpropagation algorithm known for its simplicity and which is frequently used in deep learning. Next we introduce a natural extension of the gradient system which is done by replacing the weights of individual neurones within the output layer by their nonlinear outputs. That brings more complexity to the iterative method, since the number of parameters rises considerably, but at the same time, the training data becomes built up into network in a quite natural way. The so obtained generalised gradient system is later converted to the observer one (see [6]). The aim is to turn the constant level of known first integrals into the attractor set. We will explain how the Euler iterative method, applied to the observer system, and called overfly algorithm, is involved in achieving of convergence to the global minimum of the cost function. Sections 3 and 4 discuss the applications of this algorithm in training of 1–layer and multilayer networks. The objective is to put forward an explanation of how to expand the backpropagation algorithm to its overfly version via modifying the weights updating procedure only for the first network's layer. In Section 5 we provide concrete numerical examples to illustrate the efficacy of the overfly algorithm in training of some particular neural networks.

2. Neural network without hidden layers

In this section we give an elementary algebraic description of the simplest no hidden layer neural network called also a *perceptron* (see [11]).

We define the *sigmoid* function

$$\sigma(t) = \frac{1}{1+e^{-t}}, \quad t \in \mathbb{R}\,, \tag{2.1}$$

as a particular solution of the *logistic* algebraic differential equation:

$$\sigma'(t) = \sigma(t)(1-\sigma(t))\,. \tag{2.2}$$

In particular, $\sigma : \mathbb{R} \to (0,1)$ is increasing and rapidly convergent map as $t \to \pm\infty$.

Let $X \in \mathbb{R}^n$ and $A \in \mathbb{R}^n$ be two vectors called respectively *weight* and *input* ones . The analytic map $f : \mathbb{R}^n \to (0,1)$ defined by

$$f_X : A \mapsto \sigma(< A, X >), \tag{2.3}$$

is called a no hidden layer neural network.

Let

$$(A^i, p_i), \quad 1 \leq i \leq N, \tag{2.4}$$

be the *training set* of (2.3) containing N input data vectors $A^i \in \mathbb{R}^n$ and corresponding scalar output values $p_i \in (0,1)$. We want to determine the weight vector X so that the N values $f_X(A^i)$ match outputs p_i as better as possible. That can be achieved by minimising the so called *cost* function

$$E(X) = \frac{1}{2} \sum_{k=1}^{N} (p_k - f_X(A^k))^2, \tag{2.5}$$

or, after the substitution of (2.3):

$$E(X) = \frac{1}{2} \sum_{k=1}^{N} (p_k - \sigma(< A^k, X >))^2. \tag{2.6}$$

In general, $E : \mathbb{R}^n \to (0,1)$ is not coercive and not necessarily convex map.

To apply the gradient descent method one considers the following system of differential equations

$$X' = -\nabla E(X). \tag{2.7}$$

Since E is always decreasing along the trajectories of (2.7), it is natural to solve it starting from some initial point $X_0 \in \mathbb{R}^n$ and use $X(t), X(0) = X_0$ to minimise E. The solution X can converge (in the ideal case) to the global minimum of E or, in the less favourable case, $||X(t)|| \to +\infty$ or X converges to local minima or saddle points.

The *backpropagation* method [11] for a neural network can be viewed as the Euler numerical method [13] of solving of a gradient system (2.7).

Here one approximates the time derivative by its discrete version

$$X'(t) \approx \frac{X(t+h) - X(t)}{h}, \tag{2.8}$$

for some small step $h > 0$ so that the approximative solution of (2.7) $\bar{X}_k \approx X(t_k)$ at time $t_k = kh$ can be obtained by iterations:

$$\bar{X}_{k+1} = \bar{X}_k - h\nabla E(\bar{X}_k), \quad \bar{X}_0 = X_0, \quad k \geq 0. \tag{2.9}$$

We write (2.7) in a more simple algebraic form by introducing the additional variables

$$M_i = \sigma(< A^i, X >), \quad i = 1, \ldots, N, \tag{2.10}$$

representing the nonlinear outputs of the network for N given inputs A^i of the training set. Using the equations (2.7) to compute the derivatives M_i', one obtains the following system of N differential equations

$$M_i' = M_i(1-M_i)\sum_{j=1}^{N}(p_j - M_j)M_j(1-M_j)G_{i,j}\,, \tag{2.11}$$

with $G = G_{i,j} =< A^i, A^j >$ – the $N \times N$ symmetric Gram matrix. We call (2.11) the *generalised gradient system*.

Let D be $n \times N$ matrix defined by $D = (A^1, \ldots, A^N)$. Then $G = D^T D$ and, as known from the elementary linear algebra: $\mathrm{rank}(G) = \mathrm{rank}(D)$ and $\mathrm{Ker(G)} = \mathrm{Ker(D)}$. Since the number of training vectors N usually exceeds the total number of weights n of the network, we can assume that $N > n$.

Thus, since $\mathrm{rank}(G) \leq n$, we have $\dim(\mathrm{Ker}(G)) \geq N - n > 0$.

Let $C = (C_1, \ldots, C_N)^T \in \mathrm{Ker}(G)$ be a non–zero vector from the null space of G and $I_N = (0,1)^N = (0,1)\times\cdots\times(0,1) \subset \mathbb{R}^N$. As seen from the equations (2.11), I_N is invariant under the flow of the system. Indeed, $M_i = 0$ and $M_i = 1$ are invariant hypersurfaces.

Theorem 2.1. *The function*

$$I_C = \sum_{k=1}^{N} C_k\, ln\left(\frac{M_k}{1-M_k}\right), \quad M = (M_1, \ldots, M_N)^T \in I_N\,, \tag{2.12}$$

is a real analytic first integral of the system (2.11).

There exists $p = N - \dim(\mathrm{Ker}(D)) > 0$ *functionally independent first integrals of the above form.*

Proof. The first statement can be checked straightforwardly by derivation of (2.12) using (2.11). We notice that if $0 < M_i < 1$ then $M_i/(1-M_i) > 0$. Thus, one has the real analyticity property of I_C. The linearity and functional independency of I_C, $C \in \mathrm{Ker}(D)$ follow directly from the definition (2.12). □

In the rest of the paper we will always assume that $\mathrm{rank}(D) = n$ i.e the set D contains sufficiently many independent vectors.

Let $C^1, \ldots, C^p$, $p = N - n$ be the basis of $\mathrm{Ker}(D)$. Using the vector notation

$$F(M) = \left(ln\left(\frac{M_1}{1-M_1}\right), \ldots, ln\left(\frac{M_N}{1-M_N}\right)\right)^T, \quad M = (M_1, \ldots, M_N)^T\,, \tag{2.13}$$

the family of the first integrals given by Theorem 2.1 can be written simply as

$$I_{C^i}(M) =< C^i, F(M) >, \quad i = 1, \ldots, p\,. \tag{2.14}$$

Let $H : I_N \to \mathbb{R}^p$, $I_N = (0,1)^N = (0,1) \times \cdots \times (0,1) \subset \mathbb{R}^N$ be the map defined by

$$H(M) = (I_{C^1}(M), \ldots, I_{C^p}(M))^T . \tag{2.15}$$

Lemma 2.1. *$H : I_N \to \mathbb{R}^p$ is a submersion.*

Proof. This follows directly from the fact that $C^1, \ldots, C^p$ are linearly independent vectors and (2.14). □

Thus, for all $y \in \mathbb{R}^p$ the set $\Gamma_y = I_N \cap H^{-1}(y)$ is a n–dimensional invariant manifold for the system (2.11).

Lemma 2.2. *Γ_0 is diffeomorphic to $\mathbb{R}^n$.*

Proof. Let $X \in \mathbb{R}^n$. We define the map $\Phi : \mathbb{R}^n \to \mathbb{R}^N$ by

$$\Phi(X) = (\sigma(< A^1, X >), \ldots, \sigma(< A^N, X >)^T . \tag{2.16}$$

Then, $I_{C^i}(\Phi(X)) = \sum\limits_{j=1}^{N} C_{ij} < A^j, X >=< \sum\limits_{j=1}^{N} C_{ij}\, A^j, X >= 0$ and so $\Phi : \mathbb{R}^n \to \Gamma_0$.

To show that ϕ is invertible, let us fix $M \in \Gamma_0$. Since $\sigma : \mathbb{R} \to (0,1)$ is one to one, there exists unique vector $Z = (Z_1, \ldots, Z_N)^T \in \mathbb{R}^N$, such that $M_i = \sigma(Z_i)$, for $i = 1, \ldots, N$ and

$$< C^i, Z >= 0, \quad i = 1, \ldots, p\,, \tag{2.17}$$

because $F(M) = Z$ by substitution into (2.13).

We are looking now for the solution $X \in \mathbb{R}^n$ of the linear system $< A^i, X >= Z_i$, $i = 1, \ldots, N$ which can be written in the vector form as $A^T X = Z$. The linear map $\phi : \mathbb{R}^n \to \mathbb{R}^N$, $\phi(X) = A^T X$ has $\mathrm{rank}(\phi) = n$. Moreover, $\mathrm{Im}(\phi) = \mathrm{Ker}(D)^\perp$ where orthogonality is defined by the scalar product $<,>$. Indeed, $\mathrm{Im}(\phi) \subset \mathrm{Ker}(D)^\perp$, by the direct verification, and $\dim(\mathrm{Im}(\phi)) = \dim(\mathrm{Ker}(D)^\perp)$ by the rank–nullity theorem. Hence, the map $\phi : \mathbb{R}^n \to \mathrm{Ker}(D^\perp)$ is a linear bijection and the linear equation $A^T X = Z \iff \phi(X) = Z$ admits the unique solution X since $Z \in \mathrm{Ker}(D)^\perp$ as follows from (2.17). The proof is done. □

The system (2.11) can be written in the vector form as $M' = V(M)$ where V is a complete in I_N vector field (I_N is a bounded open invariant set). Let $\epsilon > 0$ and

$$U_\epsilon = \{M \in I_N \,:\, r(M) = ||H(M)|| \le \epsilon\}\ , \tag{2.18}$$

be the ϵ–neighbourhood of Γ_0. Together with (2.11), consider the following *observer* system

$$M' = W(M) = V(M) + P(M), \quad M \in I_N\,, \tag{2.19}$$

where

$$P(M) = -k\Pi(M)\tilde{R}F(M), \quad \tilde{R} = \Theta R^{-1}\Theta^t, \quad R = \Theta^t\Theta\,. \tag{2.20}$$

. Here, $\Theta = (C^1, \ldots, C^p)$, $\Theta \in M_{p,N}(\mathbb{R})$ and

$$\Pi(M) = \operatorname{diag}\left(M_1(1-M_1), \ldots, M_N(1-M_N)\right)\,. \tag{2.21}$$

The matrix R is invertible and positive definite since $\operatorname{rank}(\Theta) = N - n$. Thus, the vector field P is well defined in I_N.

Theorem 2.2. *Let $M_0 \in I_N$ and $t \to M(t)$ be the solution of the observer system* (2.19) *with the initial condition $M(0) = M_0$. Then*

$$r(M(t)) = r(M_0)e^{-kt}, \quad t \geq 0\,, \tag{2.22}$$

with r defined in (2.18). *In particular $\lim_{t\to+\infty} r(M(t)) = 0$ and U_ϵ is invariant set containing Γ_0 as attractor.*

Proof. Firstly, we write the H introduced in (2.15) in the compact matrix form

$$H(M) = \Theta^t F(M)\,.$$

We follow now the idea of the proof of Main Lemma from [6], p. 377. and derive r^2 with respect to time along the solution of (2.19) to obtain a simple differential equation:

$$\frac{dr^2(M(t))}{dt} = -2kr^2(M(t)), \quad r^2(M(0)) = r(M_0)\,, \tag{2.23}$$

which can be easily solved to get (2.22). □

We notice that our choice of the term P in (2.19) is different from one proposed in [6].

Lemma 2.3. *The function*

$$E(M) = \frac{1}{2}\sum_{i=1}^{N}(p_i - M_i)^2\,, \tag{2.24}$$

is a Lyapunov one and verifies $\dfrac{dE(M(t))}{dt} \leq 0$ for every solution $t \mapsto M(t)$, $M(0) \in I_N$ of (2.11).

Proof. It is sufficient to derive L and to use the positiveness of the Gram matrix $G = D^T D$. □

Now we shall explain the role of the observer system (2.19) in the problem of minimisation of the cost function (2.5).

Firstly, while using the standard gradient descent method, instead of dealing with the system (2.7), one can solve the observer equations (2.19) with some initial condition

$M(0) \in \Gamma_0$ and use then Lemma 2.2 to compute X as corresponding to $M(t)$ for some sufficiently large $t > 0$. It is well known that applying the Euler method (2.8) to solve (2.7), i.e following the conventional backpropagation algorithm, leads to accumulation of a global error proportional to the step size h. At the same time, the numerical integration of the observer system (2.19), as due to the existence of the attractor set Γ_0, is much more stable numerically since the solution is attracted by the integral manifold Γ_0 (see [6] for more details and examples).

Second improvement brought by the observer system (2.19) is more promising. Imagine we start integration of (2.19) with the perturbed initial condition $M(0) \in U_\epsilon$, $M(0) \notin \Gamma_0$ for some $\epsilon > 0$. Then, according to Theorem 2.2, $M(t) \to \Gamma_0$, $t \to +\infty$ and as follows from Lemma 2.3, $t \mapsto E(M(t))$ will be decreasing function of $t > 0$ in a neighbourhood of Γ_0 since $P = 0$ on Γ_0. That can be seen as a coexistence of the local dynamics of the observer system in U_ϵ, pushing M to the equilibrium point $M_i = p_i$, $i = 1, \ldots, N$ of (2.7) and the dynamics of the gradient system (2.7) on Γ_0 forcing M to approach the critical points set (see Figure 3).

One can suggest that this kind of double dynamics increases considerably the chances of convergence to the global minimum of the cost function (2.5). We call *overfly* the training of the neural network (2.3) done by solving the observer system (2.19) with help of the Euler first–order method starting from some initial point $M(0) \in U_\epsilon \setminus \Gamma_0$.

3. The 1–hidden layer network case

In this section we describe the generalised gradient system of differential equations appearing in the supervised backpropagation training of a 1–hidden layer network. As in the previous section, let $A \in \mathbb{R}^n$ belongs to the training set (2.4). Let $Y^1, \ldots, Y^m \in \mathbb{R}^n$ be m weight vectors of the hidden layer and $X \in \mathbb{R}^m$ is the weight vector of the output layer.

The 1–hidden layer neural network is a real analytic map $f_{Y,X} : \mathbb{R}^n \to (0, 1)$ defined as follows

$$f_{Y,X}(A) = \sigma(< \pi_Y(A), X >)\,, \tag{3.1}$$

where $\pi_Y(A) = (\sigma(< A, Y^1 >), \ldots, \sigma(< A, Y^m >))^T$ are the outputs of the first layer. We want to minimise the same cost function

$$E(Y, X) = \frac{1}{2} \sum_{i=1}^{N} (p_i - f_{Y,X}(A^i))^2\,, \tag{3.2}$$

where (A^i, p_i), $i = 1, \ldots, N$ is the training set. To solve the optimisation problem one can define the gradient system analogous to (2.7) with respect to the vector variables Y^i and

X:

$$Y^{i\prime} = -\nabla_{Y^i} E, \quad X' = -\nabla_X E, \quad 1 \leq i \leq m\,. \tag{3.3}$$

Let us introduce the following scalar variables:

$$\Omega_{jk} = \sigma(< A^j, Y^k >)\,. \tag{3.4}$$

The function (3.2), expressed in new variables, takes the following form

$$E(\Omega, X) = \frac{1}{2}\sum_{i=1}^{N}(p_i - \sigma(<\Omega^i, X>))^2, \quad \Omega^i = (\Omega_{i1}, \ldots, \Omega_{im})^T\,. \tag{3.5}$$

The differential equations describing the generalised gradient system for the neural network (3.1) are obtained by derivation of (3.4) with help of (3.3):

$$\begin{cases} \Omega'_{ik} = m_{ik}(\Omega, X) = \Omega_{ik}(1-\Omega_{ik})X_k \sum\limits_{j=1}^{N}(p_j - \omega_j)\omega_j(1-\omega_j)\Omega_{jk}(1-\Omega_{jk})G_{ij}\,, \\ X' = -\nabla_X E = \sum\limits_{i=1}^{N}(p_i - \omega_i)\omega_i(1-\omega_i)\Omega^i, \quad \omega_i = \sigma(<\Omega^i, X>)\,, \end{cases} \tag{3.6}$$

where $G_{ij} =< A^i, A^j >$ is the Gram matrix defined by the training set (2.4).

The next theorem is a generalisation of Theorem 2.1. Let $r = \dim(\mathrm{Ker}(G))$ and $\mathrm{Ker}(G) = \mathrm{Span}(C^1, \ldots, C^r)$, $C^j = (C_{j1}, \ldots, C_{jr})^T$.

Theorem 3.1. *The generalised gradient system* (3.6) *admits* rm *functionally independent first integrals*

$$I_{C^j,k}(\Omega) = \sum_{i=1}^{N} C_{ji}\, ln\left(\frac{\Omega_{ik}}{1-\Omega_{ik}}\right), \quad 1 \leq j \leq r, \quad 1 \leq k \leq m\,. \tag{3.7}$$

The cost function E *defined by* (3.5) *is a Lyapunov function for* (3.6)

Proof. One verifies directly that $I_{C^j,k}$ is a first integral of (3.6) by simple derivation. A rather tedious but elementary calculation shows that $E(\Omega(t), X(t))' \leq 0$ along the solutions of (3.6) (see also Theorem 4.1 for the general proof). □

The observer system, analogous to (2.19), written for the generalised gradient system (3.6), can be obtained straightforwardly by replacing the first equation of (3.6) with

$$\Omega' = U(\Omega, X) + P(\Omega), \quad X' = -\nabla_X E, \quad 1 \leq i \leq N, \quad 1 \leq k \leq m\,, \tag{3.8}$$

where the additional term P is defined in similar to (2.20) way with help of the first integrals defined by Theorem 3.1.

Indeed, let $K = (K_{ij})_{1\leq i\leq N, 1\leq j\leq m}$ and $S = (S_{ij})_{1\leq i\leq N, 1\leq j\leq m}$ are two matrices defined by

$$K_{ij} = \Omega_{ij}(1-\Omega_{ij}), \quad S_{ij} = ln\left(\frac{\Omega_{ij}}{1-\Omega_{ij}}\right)\,. \tag{3.9}$$

To prove the result similar to Theorem 2.2 one can define P in (3.8) as follows

$$P = -kK \circ (\tilde{R}S)\,, \tag{3.10}$$

where the constant matrix $\tilde{R}$ is the same as in (2.20) and "$\circ$" is the Kronecker matrix product.

Indeed, the first integrals defined by (3.7) can be written in a matrix form: $H(\Omega) = \Theta^t S(\Omega)$. Then, deriving $r^2(\Omega(t)) = ||H(\Omega(t))||_2^2$, where $||\cdot||_2$ is the Frobenius matrix norm, along a solution $t \mapsto \Omega(t)$ of (3.8), one gets

$$\frac{dr^2(\Omega(t))}{dt} = -2kr^2(\Omega(t))\,, \tag{3.11}$$

and so

$$r(\Omega(t)) = r(\Omega_0)e^{-kt}, \quad t \geq 0\,. \tag{3.12}$$

The practical implementation of the overfly algorithm in the 1–layer case is analogous to one described in Section 2. Instead of modifying the weights of the first layer Y^i at every step of the gradient descent, one updates the values of Ω_{ik} and X applying the Euler method to solve the observer equations (3.8).

For the sake of simplicity, we will provide below the explicit matrix form of the system (3.8) which is better adopted to numerical implementations. We introduce the following diagonal matrices:

$$\begin{aligned} &\hat{P}_\omega = \mathrm{diag}((p_1 - \omega_1)\omega_1(1-\omega_1), \ldots, (p_N - \omega_N)\omega_N(1-\omega_N))\,,\\ &\hat{X} = \mathrm{diag}(X_1, \ldots, X_m)\,, \end{aligned} \tag{3.13}$$

and the N–vector

$$P_\omega = ((p_1 - \omega_1)\omega_1(1-\omega_1), \ldots, (p_N - \omega_N)\omega_N(1-\omega_N))^T\,. \tag{3.14}$$

Let $X = (X_1, \ldots, X_m)^t$ be the m–vector of the output layer. The observer system (3.8) can be written in the following compact form

$$\begin{cases} \Omega' = K \circ (G\hat{P}_\omega K\hat{X} - k\tilde{R}S) \\ X' = \Omega^T P_\omega\,, \end{cases} \tag{3.15}$$

where $K = \Omega - \Omega \circ \Omega$.

4. General multilayer case

We want to analyse a general multilayer neuronal network with the architecture $n - l - \cdots - 1$. Here n is a number of inputs and l is the number of neurones in the very first layer. The network has only one output and in every layer the same sigmoid function (2.1) is used. The training set is defined by (2.4). Let $Y^i \in \mathbb{R}^n$, $1 \leq i \leq l$ be the weight

vectors of l neurones of the first layer. We note Z the weights of other network's layers. Let $A \in \mathbb{R}^n$ be the input vector. The generic multilayer neural network can be written as the composition of two maps:

$$f_{Y,Z}(A) = \Phi_Z \circ \pi_Y(A)\,, \tag{4.1}$$

where $\Phi_Z : \mathbb{R}^l \to (0,1)$, $\pi = (\pi_1, \ldots, \pi_l)^T \overset{\Phi_Z}{\longmapsto} \Phi_Z(\pi)$ is defined jointly by all layers different from the first one and

$$\pi_Y(A) = (\sigma(< A, Y^1 >), \ldots, \sigma(< A, Y^l >))^T\,, \tag{4.2}$$

is the output vector of the first layer.

Using the chain rule one obtains for every $k = 1, \ldots, l$:

$$\frac{\partial f_{Y,Z}}{\partial Y_{ki}} = \left\langle \nabla \Phi_Z, \frac{\partial \pi_Y}{\partial Y_{ki}} \right\rangle, \quad i = 1, \ldots, n\,, \tag{4.3}$$

where, according to (4.2),

$$\frac{\partial \pi_Y}{\partial Y_{ki}} = \sigma(< A, Y^k >)(1 - \sigma(< A, Y^k >))(\underbrace{0, \ldots, A_i}_{k}, \ldots, 0)^T\,. \tag{4.4}$$

Thus, combining together (4.3),(4.4) we obtain:

$$\frac{\partial f_{Y,Z}}{\partial Y_{ki}} = \sigma(< A, Y^k >)(1 - \sigma(< A, Y^k >))A_i \frac{\partial \Phi_Z}{\partial \pi_k}\,. \tag{4.5}$$

We can compute now the partial derivatives of the cost function

$$E(Y, Z) = \frac{1}{2} \sum_{j=1}^{N} (p_j - f_{Y,Z}(A^j))^2\,, \tag{4.6}$$

with respect to the weights of the first layer:

$$\frac{\partial E}{\partial Y^k} = -\sum_{j=1}^{N} (p_j - f_{Y,Z}(A^j))\sigma(< A^j, Y^k >)(1 - \sigma(< A^j, Y^k >))\frac{\partial \Phi_Z}{\partial \pi_k}(\pi_Y(A^j))A^j\,. \tag{4.7}$$

The equation of the gradient system corresponding to the weight vector Y^k can be written as

$$Y^{k\prime} = -\nabla_{Y^k} E = -\frac{\partial E}{\partial Y^k}\,. \tag{4.8}$$

Introducing the variables

$$\Omega_{pk} = \sigma(< A^p, Y^k >)\,, \tag{4.9}$$

called the *splitting weights*, and whose derivatives can be found with help of (4.8), we deduce from (4.7) the following differential equations

$$\Omega'_{pk} = \Omega_{pk}(1 - \Omega_{pk}) \sum_{i=1}^{N} (p_i - \Phi_Z(\Omega^i))\Omega_{ik}(1 - \Omega_{ik})\frac{\partial \Phi_Z}{\partial \pi_k}(\Omega^i)G_{ip}\,, \tag{4.10}$$

where $\Omega^i = (\Omega_{i1}, \dots, \Omega_{il})^T$.

The above equations can be written also as

$$\Omega'_{pk} = N_{pk}(\Omega, Z), \quad 1 \leq p \leq N, \quad 1 \leq k \leq l\,. \tag{4.11}$$

Indeed, $f_{Y,Z}(A^j)$ and $\dfrac{\partial \Phi_Z}{\partial \pi_k}(\pi_Y(A^j))$ are functions of Ω and Z only. Moreover, the same holds for the cost function E defined in (4.6) and its gradient $\nabla_Z E = \partial E/\partial Z$: they can be written as functions of variables Ω and Z.

Let $r = \dim(\mathrm{Ker}(G))$ be the dimension of the null space of the Gram matrix $G_{i,j} =< A_i, A_j >$ and $\mathrm{Ker}(G) = \mathrm{Span}(C^1, \dots, C^r)$. We note $C^i = (C_{i1}, \dots, C_{iN})^T$.

Theorem 4.1. *Let*

$$\Omega' = N(\Omega, Z), \quad Z' = -\nabla_Z E(\Omega, Z)\,, \tag{4.12}$$

be the generalised gradient system written for the multilayer network (4.1) *with the training set* (A^i, p_i), $i = 1, \dots, N$. *Then* (4.12) *admits* rl *independent first integrals of the form*

$$I_{C^j,k}(\Omega) = \sum_{i=1}^{N} C_{ji}\, ln\left(\frac{\Omega_{ik}}{1-\Omega_{ik}}\right), \quad \Omega_{ik} = \sigma(< A^i, Y^k >)\,. \tag{4.13}$$

The cost function (4.6) $E = E(\Omega, Z)$ *is a Lyapunov function for* (4.12).

Proof. It is straightforward to verify that $I_{C^j,k}$ are functionally independent first integrals of (4.12). Accordingly to (4.1), (4.2) and (4.9), the cost function (4.6), written in variables Ω, Z, is given by

$$E(\Omega, Z) = \frac{1}{2}\sum_{i=1}^{N} (p_i - \Phi_Z(\Omega^i))^2, \quad \Omega^i = (\Omega_{i1}, \dots, \Omega_{il})^T\,, \tag{4.14}$$

in view of (4.3),(4.2) and (4.9). Let $t \mapsto (\Omega(t), Z(t))$ be a solution of (4.12). Then

$$\frac{d}{dt}E(\Omega(t), Z(t)) = \left\langle \frac{\partial E}{\partial \Omega}, N \right\rangle_\Omega - \left\langle \frac{\partial E}{\partial Z}, \nabla_Z E \right\rangle_Z = \left\langle \frac{\partial E}{\partial \Omega}, N \right\rangle_\Omega - ||\nabla_Z E||_Z^2\,, \tag{4.15}$$

where $<,>_\Omega$, $<,>_Z$ are the standard scalar products defined respectively in spaces $\mathbb{R}^a$ and $\mathbb{R}^b$ where $a = pl$ is the total number of splitting weights Ω_{pk} and b is the total number of weights Z of the neural network (4.1). One writes with help of (4.11):

$$\left\langle \frac{\partial E}{\partial \Omega}, N \right\rangle_\Omega = -\sum_{i=1}^{N}\sum_{k=1}^{l} \frac{\partial E}{\partial \Omega_{ik}}\, N_{ik} = -\sum_{k=1}^{l}\left(\sum_{i=1}^{N} T_{ik} \sum_{j=1}^{N} G_{ij} T_{jk}\right), \tag{4.16}$$

where $T_{ik} = (p_i - \Phi_Z(\Omega^i))\dfrac{\partial \Phi_Z}{\partial \pi_k}(\Omega^i)\Omega_{ik}(1-\Omega_{ik})$. Since G_{ij} is a positive matrix, the last equality implies $\left\langle \frac{\partial E}{\partial \Omega}, N \right\rangle_\Omega \leq 0$. Together with (4.15) this yields that E is a Lyapunov function of (4.12). □

The observer system, defined by analogy with (2.19) for the generalised gradient system (4.12), can be written in the following form

$$\Omega' = N(\Omega, Z) + P(\Omega), \quad Z' = -\nabla_Z E(\Omega, Z)\,, \tag{4.17}$$

where the vector field P, called the *dissipation* term, is defined by the first integrals (4.13) and given by the same formula (3.10).

The overfly algorithm for neural network training, already described in previous sections, can be easily adopted to the general multilayer case. The only difference from the conventional backpropagation applied to the network (4.1), consists in replacing the weights of the first layer Y_{ij} by the splitting weights Ω_{pk}, while keeping updating the weights Z of other layers accordingly to the usual bacpropagation algorithm. At each iteration step, the evolution of parameters Ω_{pk}, Z is governed by the Euler discretisation of the observer system (4.17).

5. Conclusion and numerical results

In this section we compare the usual backpropagation and the overfly methods for some particular neural networks. We start by a simple no hidden layer case (2.3).

We put $n = 1$ and $X = x \in \mathbb{R}$. Let $N = 5$ and the input input values are defined by

$$T = [79/100, -9/20, 7/10, -9/50, -19/25]\,, \tag{5.1}$$

with the corresponding output vector p:

$$p = [-1/20, -21/25, -11/100, 61/100, -83/100]\,. \tag{5.2}$$

The couple (T, p) defines the training set (2.4).

Analysing the equation $E'(x) = 0$, with E defined in (2.5), one calculates, with help of Maple's 10 RootFinding routine, two local minima A and B (see Figure1) of the cost function E in points $x_A = 2.510$, $E(x_A) = 1.967$ and $x_B = 6.067$, $E(x_B) = 1.966$ with B being the global minimum of E.

The gradient system (2.7) was solved using the Euler method (2.9) with $h = 1$ with the initial point $x(0) = 3$. After $d = 3000$ iterations one obtains $x = x_d = 2.510$ with $E(x_d) = 1.967$ and the backpropagation network converges to the local minimum A.

To calculate the vector $M(0)$, corresponding to $x(0)$, one can apply Lemma 2.2 to find

$$M(0) = [0.879, 0.244, 0.853, 0.389, 0.129]^T \tag{5.3}$$

Now, following the overfly approach, we consider the observer system (2.19) with $k = 0.002$ and initial conditions $M(0) + \tilde{M}$ with the perturbation vector $\tilde{M}$ defined by

$$\tilde{M} = [0.01, 0.01, 0.01, 0.01, 0.01]^T . \tag{5.4}$$

The Euler method, applied to (2.19) with $h = 1$ provides after $\delta = 3000$ iterations the value $x = \tilde{x}_\delta = 6.085$ with $E(\tilde{x}_\delta) = 1.966$. Since, $\tilde{x}_\delta$ is sufficiently close to x_B we conclude that the overfly network converges to the global minimum B rather than to the local one A. So, the benefits of the overfly training are immediately visible.

We have tested numerically the overfly method for a $4-2-1$ neural network (3.1). It has 4 inputs and 1 hidden layer with 2 neurones ($n = 4, m = 2$). Both hidden and output layer have biases. The input data set has $N = 10$ entries arranged into the following 4×10 matrix $A = [A^1 \ldots, A^{10}]$:

$$A = \begin{bmatrix} 0.234 & -0.316 & -0.746 & 0.064 & 0.124 & 0.894 & -0.786 & -0.076 & 1.044 & -0.436 \\ -0.385 & -0.835 & 0.015 & 0.365 & -0.935 & 0.135 & 0.335 & 0.505 & 0.495 & 0.305 \\ 0.764 & 0.594 & 0.684 & -0.946 & 0.024 & -0.196 & -0.596 & 0.534 & -0.436 & -0.426 \\ -1.014 & -0.074 & 0.346 & 0.876 & -0.354 & -0.184 & -0.174 & -0.254 & 0.266 & 0.566 \end{bmatrix} \tag{5.5}$$

The columns of A were chosen randomly and have zero mean. The output target vector $p \in \mathbb{R}^{10}$ is of the form

$$[0.301, 0.30001, 0.30002, 0.30013, 0.30004, 0.30005, 0.30006, 0.30007, 0.30008, 0.30009] , \tag{5.6}$$

and corresponds to a highly deviated data set. In particular:

$$\frac{p_1 - p_2}{p_3 - p_2} = 99 \quad \text{and} \quad \frac{p_6 - p_5}{p_5 - p_4} = 1 . \tag{5.7}$$

Firstly, the standard $4-2-1$ neural network (3.1) was trained on the above data set using usual backpropagation method (BM) with randomly chosen in the interval $[-1, 1]$ weights Y and X. The number of iterations was $d = 1500$ with the step size $h = 0.1$.

Then, the overfly algorithm was applied, as described in Section 3, with randomly chosen initial splitting weights $\Omega_{ij} \in (0, 1)$, same X and the dissipation parameter $k = 0.01$. The observer system (3.15) was solved by Euler method with the same step size $h = 0.1$ and using the same iteration number $d = 1500$. At each iteration we computed the cost function value for both methods: using the formula (3.2) for BM and the expression (3.5) for the overfly method (OM). The final cost value, after d iterations for BM, was $E_{BM} = 0.588 \cdot 10^{-3}$ and for OM it was $E_{OM} = 3.499 \cdot 10^{-7}$ with the ratio $E_{BM}/E_{OM} \approx 146$. Thus the overfly algorithm significantly outperforms the conventional backpropagation for this particular problem. The Figure 2 contains graphs of both cost functions in the logarithmic scale. We notice that our example is quite generic one since our numerical experiments show that statistically OM gives more precise results than BM for the large deviation output data sets.

We notice that there is an obvious resemblance between conventional backpropagation and overfly approaches. Below we summarise briefly the principal steps of the proposed method.

Step 1: Splitting. Assuming that the training data (A^i, p_i) is given, firstly, it is necessary to compute the generating vectors of the null–space of the matrix $D = (A^1, \ldots, A^N)$ i.e determine $\mathrm{Ker}(G)$. Secondly, one introduces Nl splitting weights (4.9) to replace nl weights of neurones of the first layer. In practice, the number N of training examples can be considerably larger than the input size of the network n, so the splitting brings more additional parameters to be stored in the memory.

Step 2: Dissipation. Using the vectors spanning $\mathrm{Ker}(G)$ one creates a procedure computing the dissipation term P defined by (2.20). The matrix inversion in (2.20) can be done, in the beginning, using the conjugate gradient algorithm [2] i.e in an iterative way. Indeed, the matrix R is symmetric and positive definite.

Step 3: Generalised gradient – observer: The first–order Euler iterative method is applied next to solve the observer system (4.17). The optimal choice of the step h and the constant k depends on the concrete problem. We suggest to run firstly the usual backpropagation (i.e choosing the initial value $\Omega \in \Gamma_0$) and try to improve the result using several choices of initial values for $\Omega_{ik} \in (0, 1)$ and of $k > 0$ in the overfly training. If $k = 0$ i.e then no dissipation term is present and starting with $\Omega \notin \Gamma_0$ the method can provide only the approximation of the neural network weights. But it is still worth trying: if initial values of Ω are sufficiently close to Γ_0 they will stay near Γ_0 (first integrals (4.13) are conserved) and the algorithm's complexity is greatly reduced since no dissipation is added at every iteration (no need to compute P in (4.17) at every step). Thus, the neural network can be trained in alternation with dissipation switched on and off. We notice as well that the proposed method can be easily adopted to take into account biases by introducing additional bias nodes.

Clearly, further research and more numerical evidences are necessary to confirm the benefits of the overfly algorithm. The results of our study suggest a number of new avenues for research and numerical experiments.

Acknowledgments. The study was supported by the PEPS project Sigmapad, Intelligence Artificielle et Apprentissage Automatique.

References

[1] Atakulreka A., Sutivong D., *Avoiding Local Minima in Feedforward Neural Networks by Simultaneous Learning*, Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol 4830, 2007

[2] Avriel M., *Nonlinear Programming: Analysis and Methods*, Dover Publishing, 2003

[3] Absila P.-A., Kurdyka K., *On the stable equilibrium points of gradient systems*, Systems & Control Letters Volume 55, Issue 7, July 2006, Pages 573-577

[4] Brierton J. L., *Techniques for avoiding local minima in gradient-descent-based ID algorithms*, Proc. SPIE 3066, Radar Sensor Technology II,1997

[5] Burse K., Manoria M., Kirar V.P.S., *Improved Back Propagation Algorithm to Avoid Local Minima in Multiplicative Neuron Model*, Communications in Computer and Information Science, vol 147, 2011

[6] Busvelle E., Kharab R., Maciejewski A. J., Strelcyn J.-M., *Numerical integration of differential equations in the presence of first integrals: observer method*, Appl. Math., 22, no. 3, 373–418, 1994

[7] Cetin B.C., Burdick J.W., Barhen J., *Global Descent Replaces Gradient Descent to Avoid Local Minima Problem in Learning with Artificial Neural Networks*, IEEE International Conference on Neural Networks 2, 836–842, 1993

[8] Chien-Cheng Yu, Bin-Da Liu, *A backpropagation algorithm with adaptive learning rate and momentum coefficient*, Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02, Honolulu, HI, USA, pp. 1218-1223 vol.2, 2002

[9] Fukuoka Y., Matsuki H., Minamitani H., Akimasa Ishida, *A modified back-propagation method to avoid false local minima*, Neural Networks : the Official Journal of the International Neural Network Society,11(6):1059-1072, 1998

[10] Hirsch M.W., Smale S., *Differential equations, dynamical systems, and linear algebra*, New York : Academic Press, 1974

[11] Gallant, S. I., *Perceptron-based learning algorithms*, IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191, 1990

[12] Gori M., Tesi A., *On the problem of local minima in backpropagation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume: 14, Issue: 1, 1992

[13] Hairer E., Norsett S.P., Wanner G., *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer Series in Computational Mathematics, 2nd ed., 1993

[14] La Salle J., Lefschetz S., *Stability by Liapunov's Direct Method: With Applications*, New York: Academic Press, 1961

[15] Pavelka A., Proch A., *Algorithms for initialization of neural network weights random numbers in matlab*, Proc. Control Eng., vol. 2, pp. 453-459, 2004

[16] Sprinkhuizen-Kuyper, I.G., Boers, E.J.W. *The local minima of the error surface of the 2-2-1 XOR network*, Annals of Mathematics and Artificial Intelligence 25: 107-136, 1999

[17] Sontag E.D., Sussmann H.J., *Backpropagation Can Give Rise to Spurious Local Minima Even for Networks without Hidden Layers*, Complex Systems 3, 91-106, 1989

[18] Nawi N.M., Khan A., Rehman M.Z., *A New Back-Propagation Neural Network Optimized with Cuckoo Search Algorithm*, Lecture Notes in Computer Science, vol 7971, 2013

[19] Wiggins S., *Introduction to Applied Nonlinear Dynamical Systems and Chaos*, Texts in Applied Mathematics, vol 2. Springer, New York, NY

U.M.P.A, Ecole Normale Supérieure de Lyon, 46, allée d'Italie, F69364 Lyon Cedex 07
E-mail address: atsygvin@umpa.ens-lyon.fr

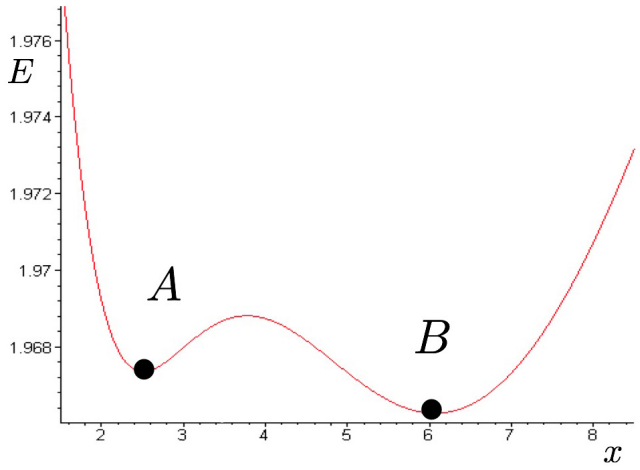

FIGURE 1. The graph of the cost function E for the training set (5.1), (5.2)

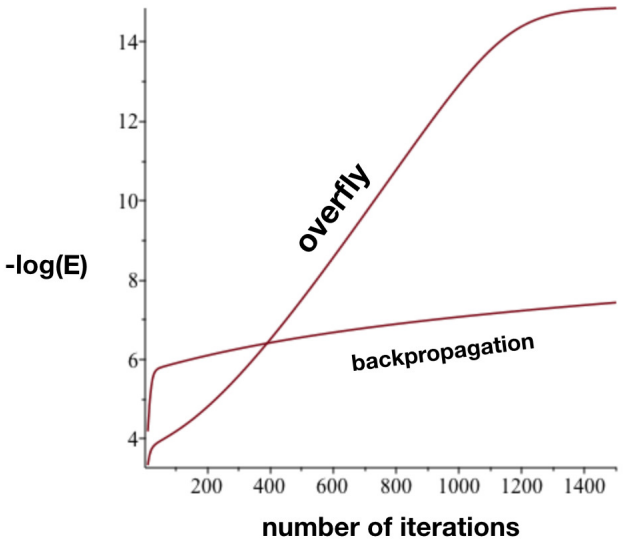

FIGURE 2. $4-2-1$ neural network, testing performance of overfly and backpropagation for the data set (5.5), (5.6)

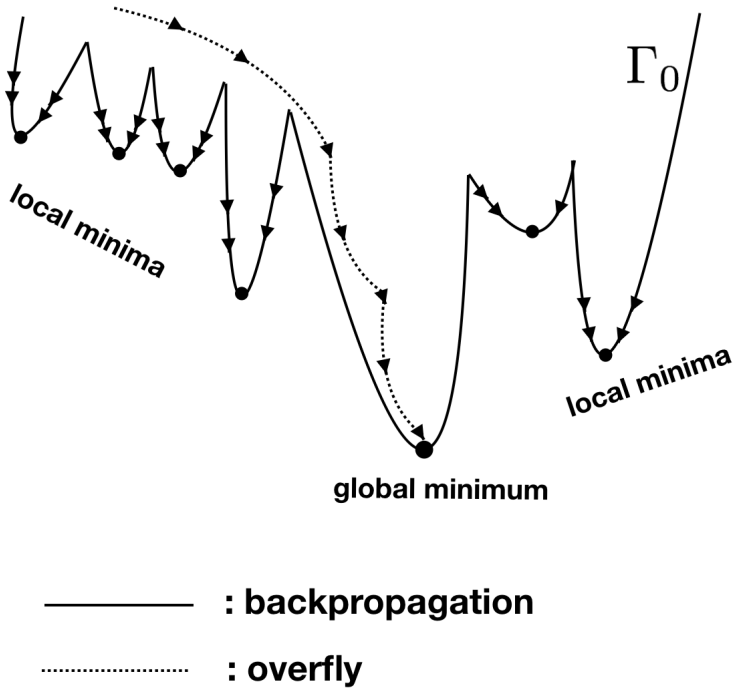
Γ_0
local minima
local minima
global minimum
: backpropagation
: overfly

FIGURE 3